John von Neumann Institute for Computing

Gb Ethernet Protocols for Clusters: An OpenMPI, TIPC, GAMMA Case Study

Stylianos Bounanos, Martin Fleury

published in

Parallel Computing: Architectures, Algorithms and Applications ,
C. Bischof, M. Bücker, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr, F. Peters (Eds.),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **38**, ISBN 978-3-9810843-4-4, pp. 397-404, 2007.
Reprinted in: Advances in Parallel Computing, Volume **15**,
ISSN 0927-5452, ISBN 978-1-58603-796-3 (IOS Press), 2008.

© 2007 by John von Neumann Institute for Computing
Permission to make digital or hard copies of portions of this work for personal or classroom use is granted provided that the copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise requires prior specific permission by the publisher mentioned above.

http://www.fz-juelich.de/nic-series/volume38

Gb Ethernet Protocols for Clusters: An OpenMPI, TIPC, GAMMA Case Study

Stylianos Bounanos and Martin Fleury

University of Essex, Electronic Systems Engineering Department
Colchester, CO4 3SQ, United Kingdom
E-mail: {sbouna, fleum}@essex.ac.uk

Gigabit Ethernet is a standard feature of cluster machines. Provision of fast network interconnects is negated if communication software cannot match the available throughput and latency. Transparent Inter Process Communication (TIPC) has been proposed as an alternative to TCP in terms of reduced message latency and system time. This study compares through low-level tests and application benchmarks the relative performance of TIPC, Open MPI, and also what improvement the GAMMA User-Level Network interface can bring compared to both of these. TIPC's system time usage is reduced compared to TCP, leading to computational gains, especially on loaded cluster nodes. GAMMA is shown to bring significant improvement in computational performance on an unloaded cluster but a more flexible alternative is Open MPI running over TIPC.

1 Introduction

Gigabit (Gb) Ethernet is a standard feature of Linux clusters and, indeed, BlueGene/L machines with this interface have headed up the Top500 supercomputer list.[a] The question arises: what is the most appropriate protocol software in terms of throughput, latency, compatibility, flexibility and so on? The traditional TCP/IP protocol suite was developed in the early 1970s for Internet applications not clusters and for relatively fast CPUs rather than network links that keep pace with the CPU. On the other hand, the Transparent Inter Process Communication (TIPC) protocol[1] promises improved short-message latency compared to the normal TCP/IP protocol suite and at data-link layer two, User-Level Network (ULN)interfaces , such as GAMMA[2], are a more direct way to improve latency and throughput. As 'Ethernot' network interfaces, Myrinet, Infiniband, QsNet, exist with custom hardware off-loaded lower layer protocol stacks, portability remains a concern, but fortunately Open MPI[3] has made it easier to integrate non-standard protocols into its component infrastructure.

This paper's contribution is a port of TIPC to Open MPI v. 1.0.2 to thoroughly benchmark the benefits of a cluster-optimized protocol compared to TCP/IP. The tests are conducted with and without background load. In turn, the paper also examines: the value of hardware off-loading of some compute-intensive TCP features onto the Network Interface Card (NIC); and whether the GAMMA Linux kernel module is a way of improving standard MPI with MPICH implementation. The communication software is evaluated by low-level metrics of performance and by a standardized set of NAS application benchmarks[4]. The cluster under test, with Gb switching and high-performance AMD processors, scales up to thirty processors; it is of a moderate but accessible size.

[a] The list is to be found at `http://www.top500.org`, checked June 2007.

2 Computing Environment

The cluster employed consists of thirty-seven processing nodes connected with two Ethernet switches. Each node is a small form factor Shuttle box (model XPC SN41G2) with an AMD Athlon XP 2800+ Barton core (CPU frequency 2.1 GHz), with 512 KB level 2 cache and dual channel 1 GB DDR333 RAM. The cluster nodes each have an 82540EM Gb Ethernet Controller on an Intel PRO/1000 NIC. This Ethernet controller allows the performance of the system to be improved by interrupt mitigation/moderation[5]. It is also possible to hardware offload routine IP packet processing to the NIC, especially TCP header Cyclic Redundancy Check (CRC) calculation and TCP segmentation.

In TCP Segmentation Offload (TSO), the driver passes 64 kB packets to the NIC together with a descriptor of the Maximum Transmission Unit (MTU), 1500 B in the case of standard Ethernet. The NIC then breaks the 64 kB packet into MTU-sized payloads. A NIC communicates via Direct Memory Access (DMA) to the CPU, over a single-width 32-bit Peripheral Component Interface (PCI) running at 33 MHz in the case of the cluster nodes. Though both the single-width and double-width (64-bit at 66 MHz) standard PCI busses should cope with 1 Gb throughput, the GAMMA optimized ULN the short-width PCI bus throughput has been found[2] to saturate at about 40 MB/s for message sizes above 2E15 B in length.

The nodes are connected via two 24 port Gigabit (Gb) Ethernet switches manufactured by D-Link (model DGS-1024T). These switches are non-blocking and allow full-duplex Gb bandwidth between any pair of ports simultaneously. Typical Ethernet switch latency is identified[6] as between 3 μs and 7 μs, whereas generally more costly Myrinet switch latency is 1 μs. The switches are unmanaged and, therefore, unable to carry "jumbo" 9000 B Ethernet frames, which would lead to an increase in communication efficiency of about 1.5% and, more significantly, a considerable decrease in frame processing overhead[7].

3 Cluster Communication Software

Transparent Inter Process Communication (TIPC) is a feature of some Linux version 2.4 kernels and is now incorporated into Linux kernel v. 2.6.16. A number of studies have identified the importance to cluster applications of latency especially[8] for short messages. Reduced latency for small-sized messages and logical addressing are two of the attractions claimed by TIPC for clusters.

Logical addressing allows realtime calculation of routes based on the zone (group of clusters), cluster, or subnet within a cluster. TIPC fuses protocol layers, which reduces the number of memory-to-memory copies within the stack, a prime cause of increased latency in protocol stacks. Connectionless and single message connection handshake remove the impediment of TCP's three-way handshake when dynamic real-time messaging is required. TIPC employs a static sliding window for flow control rather than TCP's adaptive window, which avoids complex window size calculation algorithms. In case of link congestion, message bundling up to MTU is practised. TIPC also delegates checksum error detection to the data-link layer, which without hardware-assist is a serious burden.

To compare TIPC and TCP/IP, TIPC was ported by us to Open MPI[3] and its component-based architecture. Within open MPI, the TIPC stack is accessed through the socket API and then directly to an Ethernet frame or native data-link layer protocol. Open

MPI implements the full functionality of MPI-1.2 and MPI-2. It also merges the code bases of LAM/MPI, LA-MPI, and FT-MPI, capitalizing on experience gained in their development. While open MPI is probably aimed at terascale processing at national laboratories, its flexibility allows alternative protocols to be introduced. The component architecture makes it possible to provide drivers for diverse physical layers and diverse protocols. For example, Myrinet, Quadric's qsnet, Infiniband, and Gb Ethernet are alternative physical layers, while TCP/IP and now TIPC are alternative protocols. However, Open MPI currently employs TCP for out-of-band (OOB) spawning of processes through the Open Runtime Environment (ORTE). For ease of protocol implementation, this feature was retained by us (though implementations without TCP OOB are also possible[9]).

Open MPI is organized as a series of layers of which the Byte Transfer Layer (BTL) is concerned with protocols. Open MPI supports the normal MPI point-to-point semantics, namely standard, buffered, ready, and synchronous. However, Open MPI's low-level library calls employ different types of internal protocols for point-to-point communications between Open MPI modules. The type of internal protocol depends on message size. For some message sizes, Open MPI supports software Remote DMA (RDMA).

In software RDMA, an initial message portion contains a message descriptor. If the receiver has already registered an interest in the message, then the remainder of the message is requested and transferred directly to user memory. Otherwise, the transfer only takes place when the receiver registers an interest and supplies the target user address. Function callbacks at sender and receiver in this rendezvous operation are used in the manner of Active Messages[10]. Compared to MPICH, Open MPI therefore has an extra overhead from message matching but avoids the overhead from unexpected large messages. An identifying message tag in the current implementation of Open MPI[9] is of size 16 B.

Open MPI supports three internal protocols:

1. An eager protocol for small messages, up to 64 KB in the TCP BTL. If the receiver has not registered for a message, then there is an overhead in copying the message from the receive buffers to user memory.

2. A send (rendezvous) protocol with support for RDMA for messages up to the BTL's maximum send size; parameterized as 128 KB for TCP.

3. A pipeline protocol which schedules multiple RDMA operations up to a BTL-specific pipeline depth, for larger contiguous messages. In the pipeline, memory registration is overlapped with RDMA operations.

The Genoa Active Message MAchine (GAMMA) communication layer[2] employs NIC-dependent operations through a kernel-level agent. These operations include enabling/disabling interrupts, reading the NIC status, and polling the NIC. The code is highly optimized: with inline macros rather than C functions, and receiving messages in a single and immediate operation, without invoking the scheduler. GAMMA has a 'go-back N' internal flow control protocol. GAMMA has been ported to MPI, albeit with a customized version of MPICH protocols. GAMMA also utilizes `ch_p4` for remote spawning of MPI processes.

In Linux, the New API (NAPI) can be set to reduce the rate of interrupts from the Ethernet controller by substituting clocked interrupts or polling, in a scheme originating from[11]. A by-product is that arriving packets remain in a DMA send ring[6] awaiting transfer over

the PCI bus. In the tests in Section 4, NAPI `poll rx` was activated in the Linux e1000 driver, allowing a protocol processing thread to directly poll the DMA ring for packets. In fact, by judging link congestion, NAPI is able to maintain a flat packet service rate once the maximum loss-free receive rate has been reached, reactivating NIC-generated interrupts if need be. Performance tests show a 25-30% improvement in per packet latency on a Fast Ethernet card when polling is turned on under NAPI. The 82540EM provides absolute timers which delay notification for a set period of time. The 82540EM also provides packet send/receive timers which interrupt when a link has been idle for a suitably long time. Interrupt throttling is available, allowing interrupts to be suppressed if the interrupt rate goes above a maximum threshold.

4 Results

The first set of tests considered the latency of TIPC compared against several different ways of running TCP. For the majority of the tests in this Section, the version of the Linux kernel was 2.6.16.11, and the compiler was gcc/g++ 4.0.4. The TCP-BIC version of TCP[12] is the default setting of TCP in Linux kernel 2.6.16 and this was the TCP version employed. Comparative tests between TCP-BIC and TCP New Reno previously established that no appreciable difference occurred in the outcomes of the tests, and, hence, TCP-BIC was retained. The socket send and receive buffers were 103 KB in size, this being the default and immutable size for TIPC. The default value for the Gb Ethernet NIC transmit queue of 1000 (`txqueuelen`) was also retained.

In Fig. 1, the plot with the legend 'with hw offloading' refers to off-loading of the TCP/IP checksum calculations and TCP segmentation to the NIC, TSO (Section 2). The plot with legends '. . . and with NODELAY' refer to turning off Nagle's algorithm, which causes small messages to be aggregated by TCP. The comparison was made by normalizing the TCP latencies to that of TIPC. The lower latency of the two TCP varieties without hardware offloading, may be attributable to applying TSO to small message sizes, as the PCI overhead of the segmentation template is relatively large. Another factor in hardware offloading is that, because the CPU processor runs faster than that on the Intel NIC, as the processor is repeatedly called for CRC calculation for small messages, their latency will be reduced. Of course, there is a trade-off against available processing for the application. After 64 KB, those varieties of TCP with hardware offloading have lower latency compared to TIPC, because PCI overhead is relatively reduced. In the TIPC implementation under test, the maximum message payload size was set in the TIPC header and at various points in the source code of the kernel module to 66 kB. Therefore, for message sizes above this limit, it was necessary to make two system calls, whereas the TCP socket only involves one system call. It has been stated[1] that TIPC spends 35% less CPU time per message than TCP, up to sizes of 1 KB, when the host machine is under 100% load. This is an important practical advantage, as clearly if TIPC spends relatively less time processing that time is available for computation. We were able to confirm this in Fig. 2, which shows that TIPC's throughput relative to system time (captured with the Unix utility `getrusage`) is dramatically better.

Comparing TIPC and TCP running under Open MPI, Fig. 3 shows that for message-sizes below about 16 KB, TIPC has a clear advantage in latency. There is also a sharp increase in TIPC latency at around 128 KB. TCP was able to employ an appropriate Open

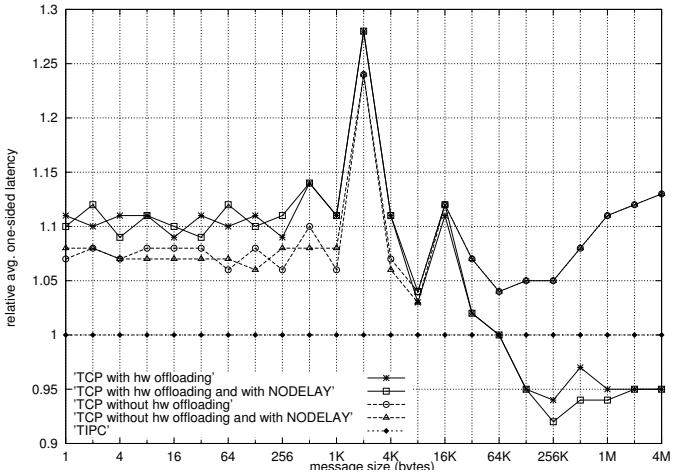

Figure 1. Comparative latency of TCP and TIPC across a range of message sizes, normalized to TIPC. (Note the log. scale on the horizontal axis.)

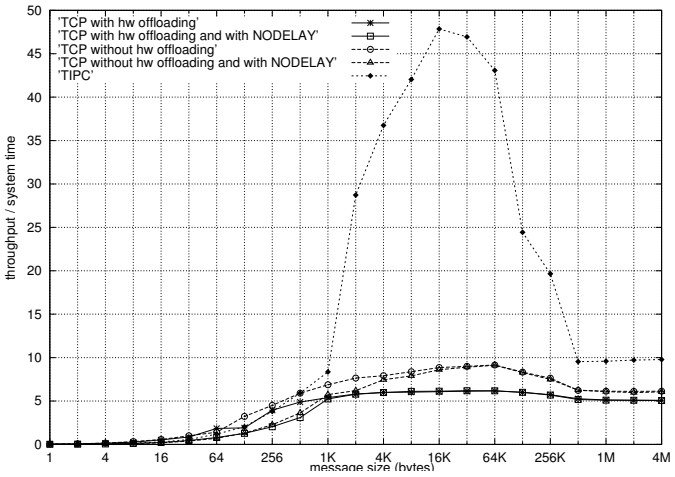

Figure 2. Comparative resource usage of TCP and TIPC across a range of message sizes. (Note the log. scale on the horizontal axis.)

MPI internal protocol, depending on message size. The same message parameterizations were retained in the TIPC BTL as in the TCP BTL. In the TIPC implementation under test, the maximum message payload size was again internally set to 66 kB. To avoid problems with open MPI in the pipelined version of RDMA, the third internal protocol for larger messages was disabled in the TIPC BTL, and the send protocol maximum was set to 66 kB. For message sizes of 128 KB and above, TIPC performs up to three rendezvous operations (for the first eager fragment, a maximum send-sized portion, and any remainder via the

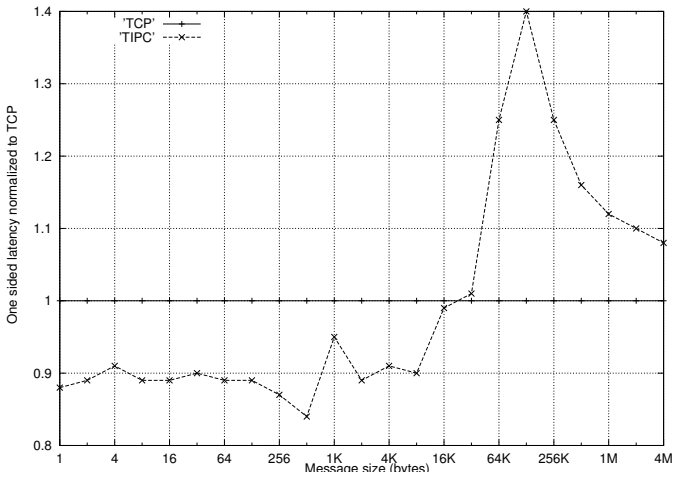

Figure 3. Comparative latency of OMPI/TCP variants and OMPI/TIPC across a range of message sizes, normalized to TCP. (Note the log. scale on the horizontal axis.)

send internal protocol), whereas the TCP BTL negotiates a single RDMA operation for the entire message. Within the latency range that TIPC has the advantage, there is a 'sweet-spot' at about 512 KB, possibly due to internal buffering arrangements.

Apart from low-level message passing tests, we also conducted application level tests. The NAS Parallel Benchmarks (version 3.2)[4] are a set of eight application kernels/programs, *viz.* CG, IS, LU, FT, MG, BT, and SP, and EP, with MPI source code. Open MPI was compiled with Fortran 77 bindings to match the NAS source code. We used NAS W class problems, as the dimension of these problems is appropriate to the cluster's processors. Accurate recording of CPU load is available through Andrew Morton's 2003 utility `cyclesoak`, which can also act as a source of load. Following from the findings on relative system time usage, two sets of tests were conducted, with and without load. In fact, the EP benchmark could also be adapted as a timewaster, as it continually creates pseudo-random numbers. Notice that as EP is CPU-bound, it only communicates at the end of its operations. In the tests, the best of five runs was taken rather than the median of five runs. This was because one of the application kernels, the IS integer sort, was found to be particularly sensitive to any system activity. For the others, the median does not differ noticeably from the best. LU and SP are computational fluid dynamics, while the others are diverse 'kernels'. For example, MG is a multigrid kernel for solving a 3D Poisson Partial Differential Equation set, being a hierarchical version with rapid convergence of the application.

Table 1 records the results. The Table also includes results for the MPICH version of MPI, with `ch_p4` being the standard MPICH method of spawning remote processes via the Unix utility rsh. The first two rows of Table 1 are taken from Table 2 of[13], also for 16 processors and W class problems. It will be seen that the message sizes are large for some of the tests, though for LU and MG they are about 1 KB and below the MTU. From the results, observe that MPICH almost always under-performs Open MPI. Noticeable gains in

	Application benchmark					
Test	CG	IS	LU	MG	SP	EP
Comms. freq. (MB/s)	36.24	22.10	5.58	20.21	19.50	0.00
Comms. freq. (# msg./s)	6103	2794	5672	13653	1657	16
Avg. msg. size (B)	5938	7909	983	1480	11768	
mpich/ch_p4	436.65	88.18	4134.11	2140.89	1698.00	140.86
OMPI/tcp	493.16	91.49	4688.89	2220.64	1792.43	141.84
OMPI/tipc	503.36	90.77	4921.43	2280.77	1812.18	140.88
tipc rel. tcp (%)	+2.07	-0.79	+4.96	+2.71	+1.10	-0.68
ch_p4 rel. tcp (%)	-11.46	-3.62	-11.83	-2.71	-3.59	-0.69
mpich/ch_p4 (loaded)	474.26	90.44	4137.57	2152	1686.01	141.55
OMPI/tcp (loaded)	494.70	91.56	4021.40	2237.23	1522.23	141.62
OMPI/tipc (loaded)	506.31	91.54	4356.74	2308.87	1574.66	141.71
ch_p4 rel. tcp (%)	-4.13	-1.22	+2.89	-3.77	+10.76	-0.05
tipc rel. tcp (%)	+2.35	-0.02	+8.34	+3.20	+3.44	+0.06

Table 1. NAS W class benchmark results (MOP/s) for sixteen processors, including relative (rel.) performance.

performance occur for the TIPC variant of Open MPI in the LU and MG application, both with shorter messages. Lastly, TIPC's relative performance compared to TCP increases when there is background load, suggesting an efficient stack.

We were also interested in the performance of GAMMA. A further set of tests were conducted in which later versions of the software were availed of. TIPC version 1.6.2, which we applied to kernel 2.6.18, removes the 66 kB send limit in the sense that the programmer no longer need make two system calls for messages over this limit, though the underlying MTU size of course remains. Table 2 shows very good relative performance of GAMMA but much weaker performance under load. At this point in time, the cause of GAMMA's weak performance under load has not been established by us despite investigations. We have not included MPICH results in Table 2 as its relative performance was weak, particularly when there was no background load. This decision was taken as we could not be sure that new software settings had resulted in disadvantaging native MPICH.

5 Conclusion

The tests comparing TCP and TIPC reveal that TIPC has very real advantages over TCP, both within and outside an OMPI environment. This is the paper's strongest conclusion, as it is shown for low-level benchmarks and for NAS application kernels. Short message latency was reduced. Moreover, the efficiency of the TIPC stack is demonstrated for nodes with high background load. The TIPC protocol has only recently been transferred to the Linux kernel and will gain as its software matures. A speculation is what would be the gain if TIPC benefited from hardware offloading to the NIC for larger messages, as TCP is able to do. The GAMMA user-level network interface (with MPI) is also capable of much improved performance over a combination of TCP and MPI. However, on a heavily-loaded machine, its computational performance may be matched by OMPI/TIPC.

	Application benchmark					
Test	CG	IS	LU	MG	SP	EP
mpich/GAMMA	876.07	140.37	7608.99	3222.95	2408.11	146.47
OMPI/tcp	521.73	94.62	4807.69	2289.76	1850.37	145.68
OMPI/tipc	524.16	94.46	5003.33	2310.14	1811.60	145.67
GAMMA rel. tcp (%)	+67.92	+48.75	+58.27	+40.75	+30.14	+0.54
tipc rel. tcp (%)	+0.47	-0.17	+4.07	+0.89	-2.10	-0.01
mpich/GAMMA (L)	56.69	18.46	1261.28	607.35	1299.29	134.67
OMPI/tcp (L)	478.62	92.90	3844.91	2276.31	1516.27	139.77
OMPI/tipc (L)	508.21	92.75	4022.34	2293.77	1503.29	140.06
GAMMA rel. tcp (L,%)	-88.16	-80.13	-67.20	-73.32	-59.87	-3.65
tipc rel. tcp (L,%)	+6.18	+0.16	+4.61	+0.77	-0.86	+0.21

Table 2. NAS W class benchmark results (MOP/s) for sixteen processors with GAMMA, including relative (rel.) performance; (L) indicates loaded benchmark.

References

1. J. P. Maloy, *TIPC: providing communication for Linux clusters*, in: Linux Symposium, vol. **2**, pp. 347–356, (2004).
2. G. Ciaccio, M. Ehlert and B. Schnor, *Exploiting gigabit ethernet for cluster applications*, in: 27th IEEE Conf. on Local Computer Networks, pp. 669–678, (2002).
3. E. Gabriel *et al.*, *Open MPI: goals, concept, and design of a next generation MPI implementation*, in: 11th Eur. PVM/MPI Users' Group Meeting, pp. 97–104, (2004).
4. D. Bailey *et al.*, *The NAS parallel benchmarks 2.0*, Tech. Rep. NAS-95-020, NASA Ames Research Center, Moffet Field, CA, (1995).
5. *Interrupt moderation using Intel gigabit ethernet controllers*, Tech. Rep., Intel Corporation, Application note AP-450, (2003).
6. A. Gallatin, J. Chase, and K. Yochum, *Trapeze/IP: TCP/IP at near gigabit speeds*, in: USENIX'99 Technical Conference, (1999).
7. R. Breyer and S. Riley, *Switched, Fast, and Gigabit Ethernet*, (Macmillan, San Francisco, 1999).
8. P. Buonadonna, A. Geweke, and D. Culler, *An implementation and analysis of the Virtual Interface Architecture*, in: Supercomputing Conference, pp. 1–15, (1998).
9. B. W. Barrett, J. M. Squyres, and A. Lumsdaine, *Implementation of Open MPI on Red Storm*, Tech. Rep. LA-UR-05-8307, Los Alamos National Lab., (2005).
10. T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser, *Active Messages: a mechanism for integrated communication and computation*, in: 19th Annual International Symposium on Computer Architure, pp. 256–266, (1992).
11. J. C. Mogul and K. K. Ramakrishnan, *Eliminating receive livelock in an interrupt-driven kernel*, ACM Transactions on Computer Systems, **15**, 217–252, (1997).
12. L. Xu, K. Harfoush, and I. Rhee, *Binary increase congestion control for fast, long distance networks*, in: IEEE INFOCOM, pp. 2514–2524, (2004).
13. K. Morimoto *et al.*, *Performance evaluation of MPI/MBCF with the NAS parallel benchmarks*, in: 6th Eur. PVM/MPI Users' Group Meeting, pp. 19–26, (1999).